Safe{Core} Protocol

[Draft]

Lukas Schor, Richard Meissner

August 2023

ABSTRACT

The *Safe{Core} Protocol* introduces an open-source interoperability protocol for modular smart contract-based accounts ("smart accounts"). It addresses challenges in the smart account ecosystem such as fragmentation, vendor lock-in and security risks. This vendor-agnostic protocol supports *Modules* like *Plugins*, *Hooks*, *Signature Validators* and *Function Handlers* which extend the functionality of *Accounts*, while *Registries* enable enforcement of standards and security properties. Future iterations aim to introduce chain abstraction, fee mechanisms, and support for new module types. The *Safe{Core} Protocol* is an ecosystem initiative to foster the long-term growth and development of smart accounts by ensuring composability, portability and security.

1. INTRODUCTION

1.1. Account Abstraction

For the Ethereum Virtual Machine (EVM) ecosystem to evolve, there needs to be a transition from externally owned accounts to smart accounts, unlocking usability and security features such as recovery mechanisms, multi-signature wallets, transaction batching, automations, gas abstraction and more. This transition is generally referred to as *Account Abstraction* and is supported by efforts such as:

- ERC-1271 is designed for smart accounts to handle off-chain functionalities. This standard provides a robust and customizable method to validate signatures, thereby enhancing the versatility of smart accounts.
- ERC-4337 attempts to standardize and abstract the account features without changes to the consensus layer. Its primary focus is to enable smart accounts to implement arbitrary transaction fee payment mechanisms.
- ERC-6900 tackles the modularity challenge in smart accounts through common interfaces, thereby fostering a foundation for efficient and reusable smart account modules.

These initiatives (and others like EIP-3074, ERC-5005, ERC-6492, EIP-7377) serve as distinct building blocks, each targeting specific aspects of smart accounts, and have previously only been conceptually connected. By explicitly integrating these initiatives, *Safe{Core} Protocol* aims to streamline the development of smart accounts, and pave the way for the widespread adoption and expansion of the smart account ecosystem.

1.2. Motivation

The *Safe{Core} Protocol* is an open-source framework for modular smart accounts that are composable, portable and secure. The protocol is inspired by the learnings from the *Safe Smart Account*, but is fundamentally vendor-agnostic and meant as an ecosystem initiative, solving the following challenges:

- **Fragmentation:** As various smart accounts are developed by vendors, tooling and applications have to prioritize which ones to be compatible with. This may result in major fragmentation of the smart account ecosystem and lack of component reuse. This degradation of both developer and user experience reduces the overall growth and utility of smart accounts.
- **Vendor Lock-In:** Externally owned accounts are a widely accepted standard, allowing portable accounts across wallets and applications. The fragmentation of smart accounts, however, may create vendor lock-in due to wallets creating proprietary account implementations that reduce portability. End users may face restrictions when trying to move their assets or interact with different services.
- **Security Risks:** Smart accounts introduce additional smart contract risk, similar to other smart contract-based systems like bridges or DeFi protocols. Without sufficient security, users will not feel comfortable leveraging the full potential of smart accounts.

To address these challenges the *Safe{Core} Protocol* aims to achieve maximum reuse and interoperability of components, retain portability, and establish stronger security properties for smart accounts. Developers adopting the *Safe{Core} Protocol* benefit from increased security, interoperability, and composability with the wider smart account ecosystem, as well as means to charge fees in the future.

2. PROTOCOL COMPONENTS

The architecture of the *Safe{Core} Protocol* is composed of several components, each with unique responsibilities and interfaces. This allows independent expansion and iteration of each component. This modularity ensures the sustainable growth and evolution of the smart account ecosystem.

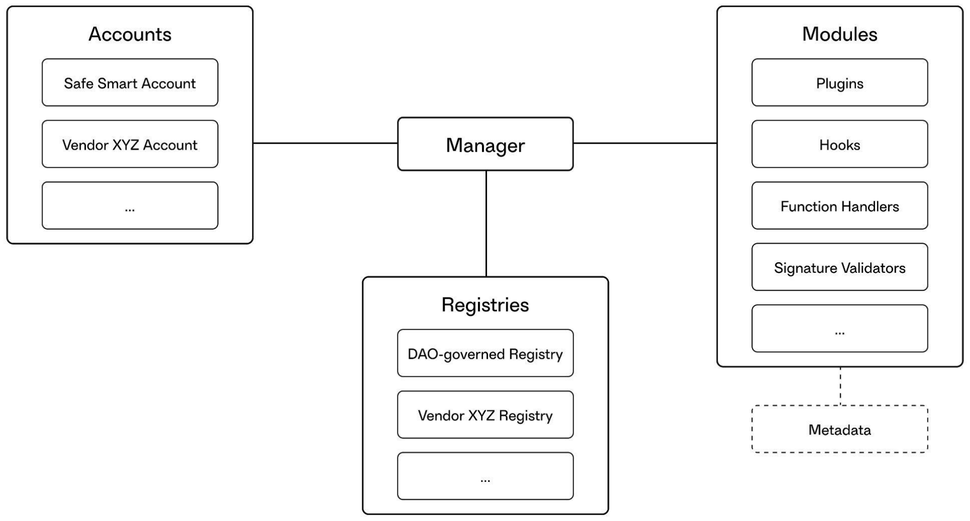

Figure 1: *Safe{Core} Protocol* architecture

2.1. Manager

At the center of the protocol is the *Manager*, which serves as a coordination point between *Accounts, Registries* and *Modules* (see *figure 1*). The *Manager* implements an abstraction layer to handle the modularity of smart accounts. Users can expand the functionality of their *Accounts* by activating a *Registry* and enabling *Modules* that are registered in the chosen *Registry* (see *figure 2*).

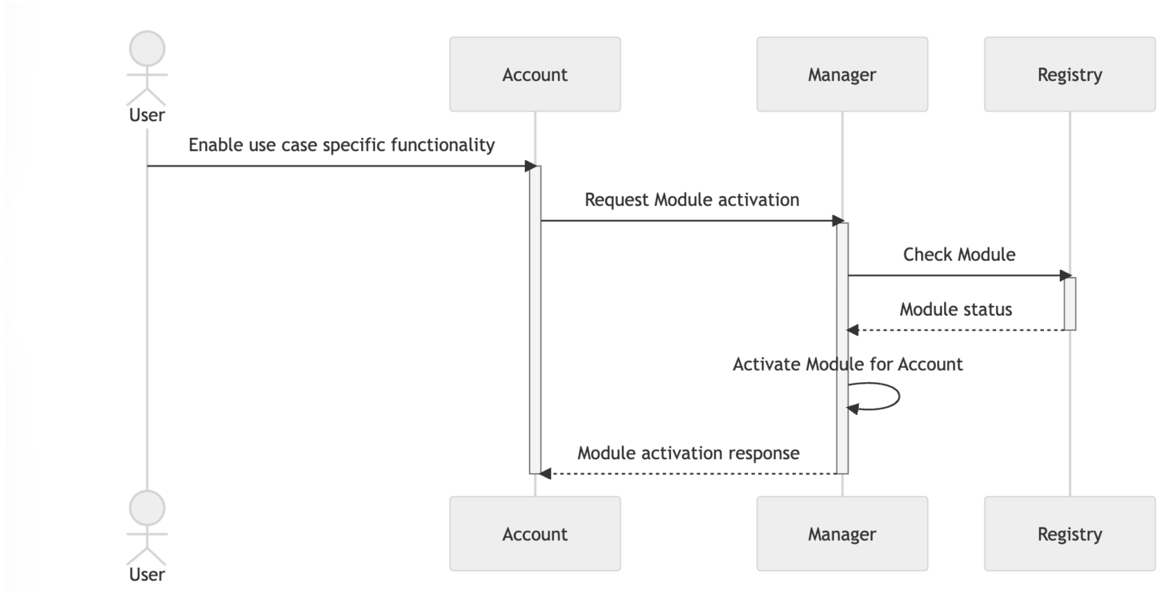
User
Account
Manager
Registry
Enable use case specific functionality
Request Module activation
Check Module
Module status
Activate Module for Account
Module activation response
User
Account
Manager
Registry

Figure 2: *Module activation flow*

The *Manager* also extends the security of smart accounts by acting as a permissioning layer, providing the ability to grant varying levels of permissions to different *Modules*, such as restricting *Modules* to perform configuration changes or *delegatecall* operations on an *Account*. Additionally, the *Manager* allows implementing different default transaction fee handling schemes, such as ERC-4337, as they may vary depending on the network.

The *Manager* is shared between different *Accounts* and activated by the user. It is non-upgradable for security reasons, but different deployments are expected over time, adding new functionality and allowing users to migrate themselves.

2.2. Accounts

Accounts are user-owned smart accounts that opt-in to the protocol by enabling the *Manager*, which can happen already during account deployment or when adding the first *Module*. The *Safe{Core} Protocol* is vendor-agnostic, meaning that it is not tied to a specific account implementation, as long as it supports the account interface defined by the *Manager*.

The initial account interface is optimized for 1.x versions of Safe Smart Account to expedite the development process and gather feedback. These learnings will be the foundation upon which the account interface is generalized. This will allow for a wider range of existing and new account implementations to be compatible with the protocol, and provide an open infrastructure piece for the ecosystem.

2.3. Modules

Modules extend the functionality of *Accounts* in different ways. Initial module types are *Plugins*, *Hooks*, *Signature Validators* and *Function Handlers*, but additional types can be added to the *Safe{Core} Protocol* over time. These module types are also assumed to be impacted by standards such as ERC-6900.

2.3.1. Plugins

Plugins allow adding any arbitrary logic to an *Account* such as recovery mechanisms, session keys and automations. These *Plugins* exist as externally deployed smart contracts that are activated in the *Manager* and act as alternative entry points to trigger account transactions (see *figure 3*). They can provide additional functionality for wallets, but can also be app-specific plugins that enable developers to build more powerful applications.

2.3.2. Hooks

Hooks add additional logic at specific points of the transaction lifecycle. The *Safe{Core} Protocol* recognizes two types of hooks:

- *Pre-Execution Hooks* run before a transaction is being executed.
- *Post-Execution Hooks* run at the end of a transaction.

Hooks enable various forms of security protections, such as allow- and deny-lists, MEV and slippage protections, risk and compliance assessments, and more.

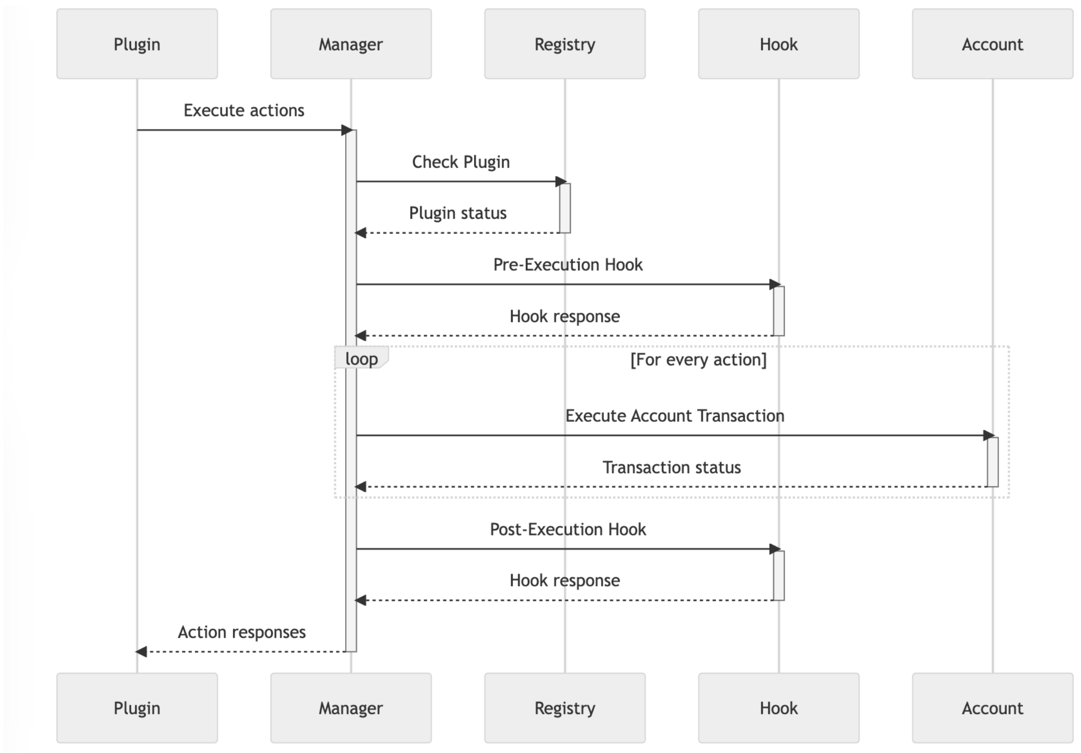

Figure 3: *Plugin execution flow using Hooks*

2.3.3. Signature Validators

Signature Validators leverage the ERC-1271 standard, enabling customizable logic for (domain-based) signature validation. *Signature Validators* incorporate off-chain functions, increasing the versatility of smart accounts. This also empowers intent-based architectures such as CowSwap's Time-Weighted Average Price (TWAP) feature.

2.3.4. Function Handlers

Function Handlers manage arbitrary function calls when an unsupported function is called in the *Account*. This enables *Accounts* to contain minimal logic and remain compatible with emerging standards, such as new token types. Like all other module types, Function Handlers have to be activated with care, as even read-only functions can impact the state of the *Account* if external contracts rely on them for authentication.

2.4. Registries

Registries play a critical role in establishing stronger security properties and enforcing standards (e.g. ERC-6900) for *Modules*. Their role is akin to a centralized repository or an app store for smart account modules, housing a comprehensive list of approved and available *Modules*. *Accounts* set the registry in the *Manager* and, as a result, can only activate *Modules* that are approved on this specific registry.

To enhance security, *Registries* can set unique inclusion/exclusion criteria and processes. For example, a *Registry* may establish automated processes to detect vulnerabilities and automatically deactivates the faulty *Modules* on *Accounts*, potentially giving users a grace-period to overrule this decision. This allows users to benefit from the increased functionality of *Modules* while maintaining high security standards.

Registries may also enable discoverability by facilitating curation methods, serving as a convenient reference point for developers and users. For example, a *Registry* may expose only those *Modules* that are supported in the user interface of a specific wallet vendor to ensure stronger security and usability properties.

2.5. Metadata

In order to build powerful tooling, wallets and applications on top of the *Safe{Core} Protocol*, developers can give additional context to *Modules* and module transactions as *Metadata*. This allows developers to add additional information such as "name", "version", "description", etc. For example, developers can include a link to a frontend widget that can be leveraged as a user interface to the *Module*. *Registries* may use information about the version of the *Module* in order to suggest updates to the user. The *Metadata* can also be used to give additional context to a transaction such as which wallet vendor facilitated it.

Metadata utilizes ERC-712 to make the information machine-readable, on-chain-verifiable and semi human-readable.

3. OUTLOOK

The above constitutes an early concept of the *Safe{Core} Protocol* with an early reference implementation being worked on. There are multiple pathways to develop the protocol further.

3.1. Fees Mechanisms

Accounts, Registries and *Modules* should be able to charge fees, such as one-time activation fees, usage-based fees or subscription fees. These fees can be taken by the developer or shared with other stakeholders, such as wallets or *Registry* maintainers. This would foster a more sustainable and vibrant smart account ecosystem by providing financial incentives to participate and contribute in the *Safe{Core} Protocol*.

3.2. Chain Abstraction

In the current architecture, the *Safe{Core} Protocol* and its components have to be deployed on each chain individually. Future versions should introduce ways for protocol components to be more easily accessed cross-chain in a trust-minimized or trussless way.

3.3. More Modules Types

Besides existing *Modules* (*Plugins*, *Hooks*, *Signature Validators, Function Handlers*), there could be additional module types that can be incorporated into the *Safe{Core} Protocol* to expand the flexibility and capabilities of smart accounts. Also existing module types could be extended, such as expanding Hooks to also support plugin-specific hooks.

4. CONCLUSION

The *Safe{Core} Protocol* pioneers a modular, open-source interoperability protocol that advances the development of smart accounts, introducing a comprehensive suite of module types for enhanced functionality. Its vendor-agnostic design ensures high component reuse and maintains account portability. *Registries* provide discoverability of *Modules*, upholds security and interoperability properties, and facilitates vulnerability responses. Looking ahead, the protocol will evolve to accommodate chain abstraction, establish fee structures, and integrate additional module types. This work demonstrates the potential of the *Safe{Core} Protocol* to support the EVM ecosystem's transition to smart accounts, improving developer and user experiences.

REQUEST FOR FEEDBACK

We invite the wider community to provide feedback to refine the *Safe{Core} Protocol*, develop a roadmap for future iterations and build the implementation in public.

A work-in-progress of the protocol specs and reference implementation can be found here, and feedback is appreciated on GitHub or in this Forum. This document will be updated to incorporate feedback.

ACKNOWLEDGEMENTS

For providing feedback on an early draft to Aaron Cook (*Safe*), Anirudh Chohan (*Safe*), Akshay Patel (*Safe*), Andre Geest (*Safe*), Auryn Macmillan (*Gnosis Guild*), Bapi Reddy (*Brahma*), Bil Loul (*Nest Wallet*), Christoph Simmchen (*Safe*), Christopher Heymann (*1kx*), Clement Bihorel (*Safe*), Daniel Fischmann (*L1D*), Daniel Partida (*Safe*), Eylon Aviv (*Collider*), Frederico Sabino (*Safe*), Ivo Georgiev (*Ambire Wallet*), Jan Baeriswyl (*very early Ventures*), John Ennis (*Safe*), Jonah Ehrlich (*Den*), Josef Gattermayer (*Ackee Blockchain*), Kurt Larsen (*Rhinestone*), Lefteris Karapetsas (*Rotki*), Marvin Lin (*Gnosis Guild*), Micah Zoltu (*serv.eth*), Nichanan Kesonpat (*1kx*), Poolpi Tako (*Yearn*), Robert Koschig (*1kx*), Sebastian Bürgel (*Hopr*), Stefan George (*Gnosis*), Steven Biekens (*Safe*), Tobias Schubotz (*Safe*), Vitaly Bahachuk (*Linen Wallet*).